🧮 Lexicographical Counting

Suppose you want to print numbers from 1 to 1000 — you could easily do that using a simple `for` loop. But to print them in **lexicographical order**, we need to use a different approach.

📘 What Is Lexicographical Order?

Lexicographical order (also known as **dictionary order**) is a way of ordering items — like strings or numbers — by comparing them **character by character**, just like words are arranged in a dictionary.

🔤 Example 1: Comparing `55` and `7`

Which one is lexicographically larger?

It's **7**, because we compare the **ASCII values** of the characters.

- First characters: `'5'` vs `'7'`
- Since `'7'` comes after `'5'`, **7 is considered greater than 55** in lexicographical order — even though numerically it's smaller.

🔤 Example 2: Comparing `10` and `101`

Let's compare the strings `"10"` and `"101"`:

- `'1'` vs `'1'` → equal
- `'0'` vs `'0'` → still equal
- `"10"` ends, but `"101"` has one more character `'1'`

So, **"10" comes before "101"** in lexicographical order.

💡 Lexicographical Order in LeetCode 386

In the problem LeetCode 386: Lexicographical Numbers, you're given an integer `n` and asked to return all numbers from `1` to `n` **in lexicographical order**.

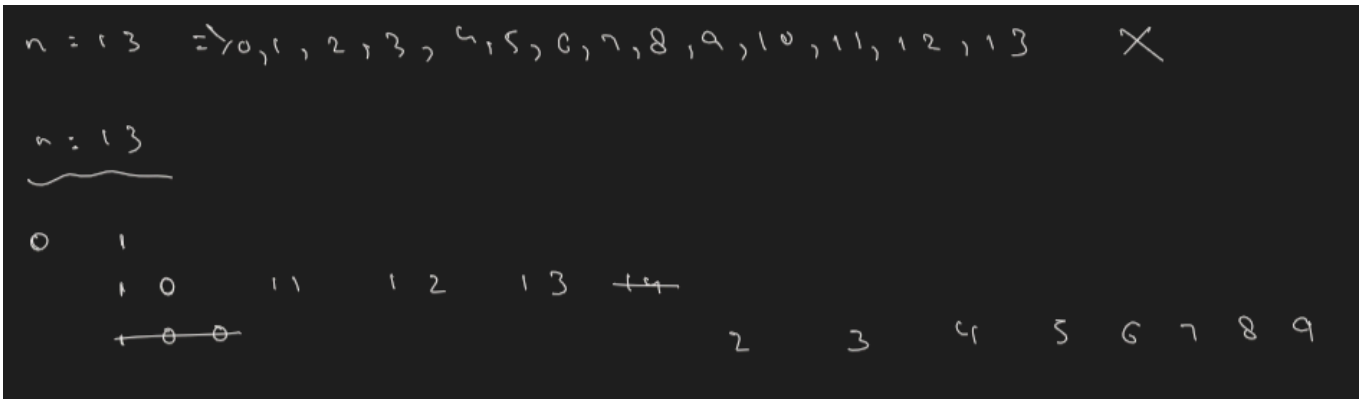
For example, if `n = 13`, the lexicographical order is:

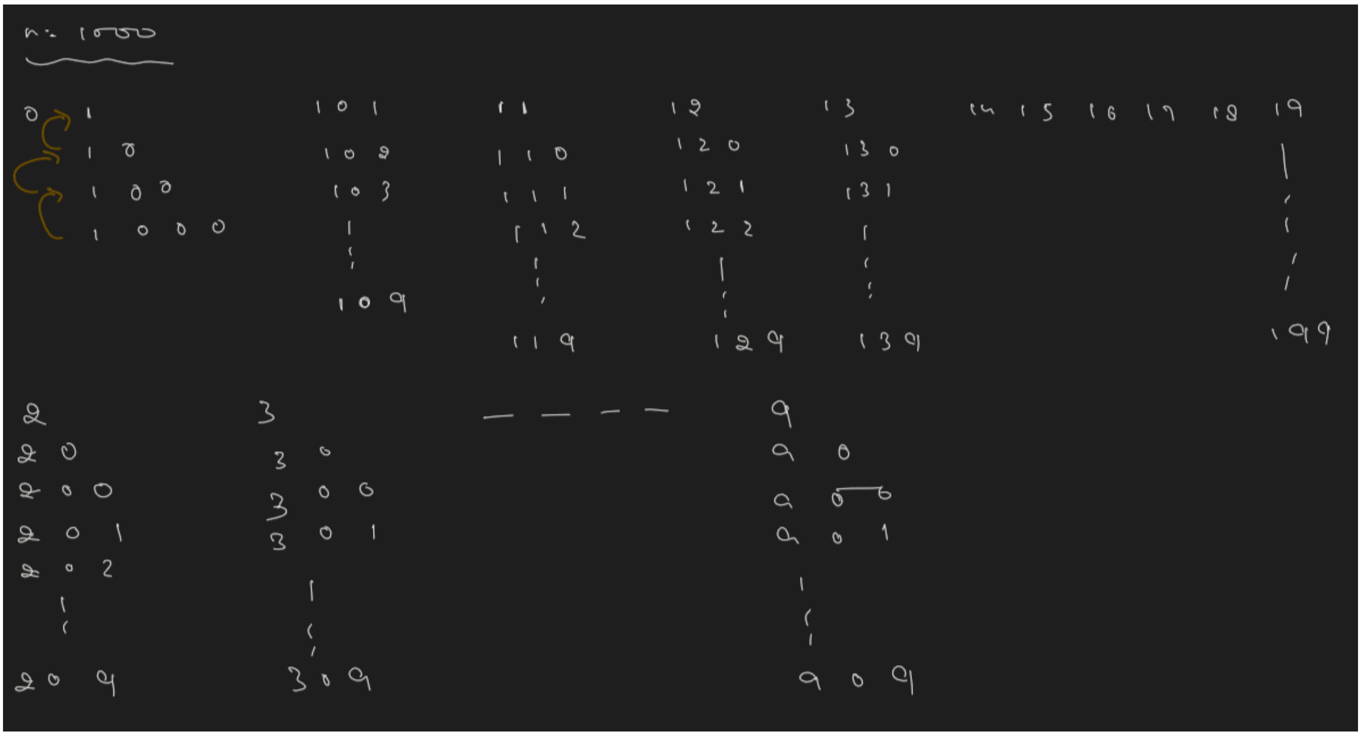
`1, 10, 11, 12, 13, 2, 3, 4, 5, 6, 7, 8, 9`

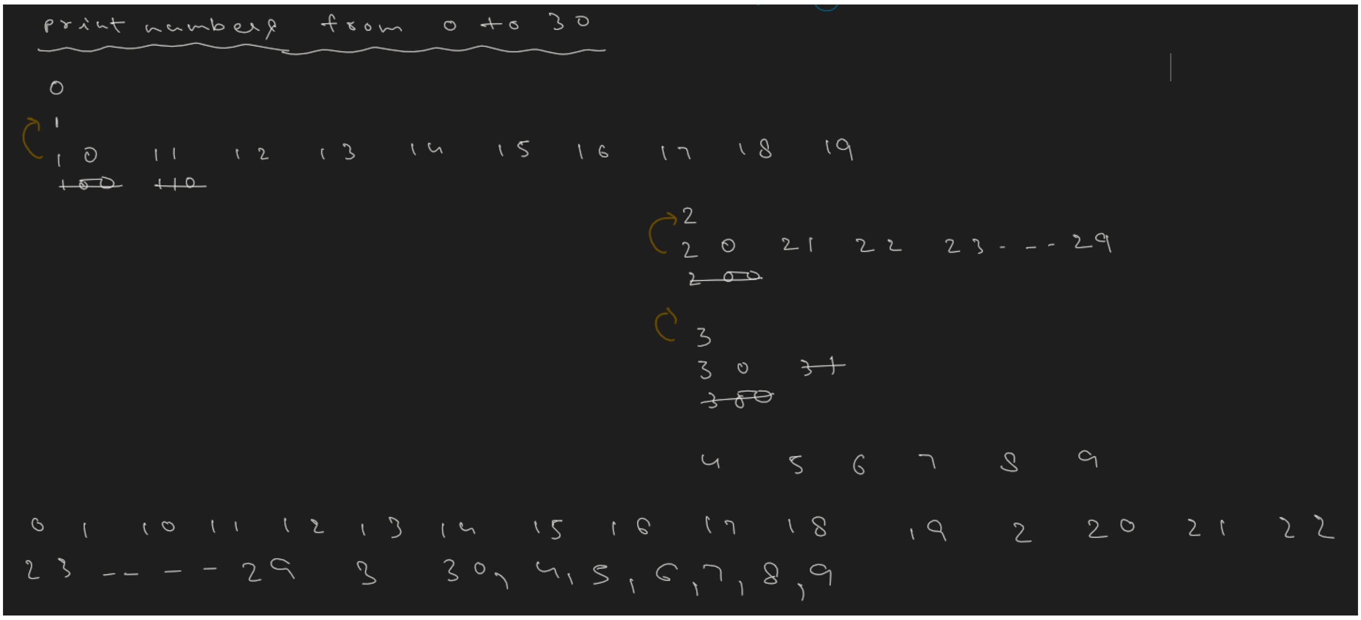
because "10", "11", … "13" all begin with "1", so they come right after "1", before "2".

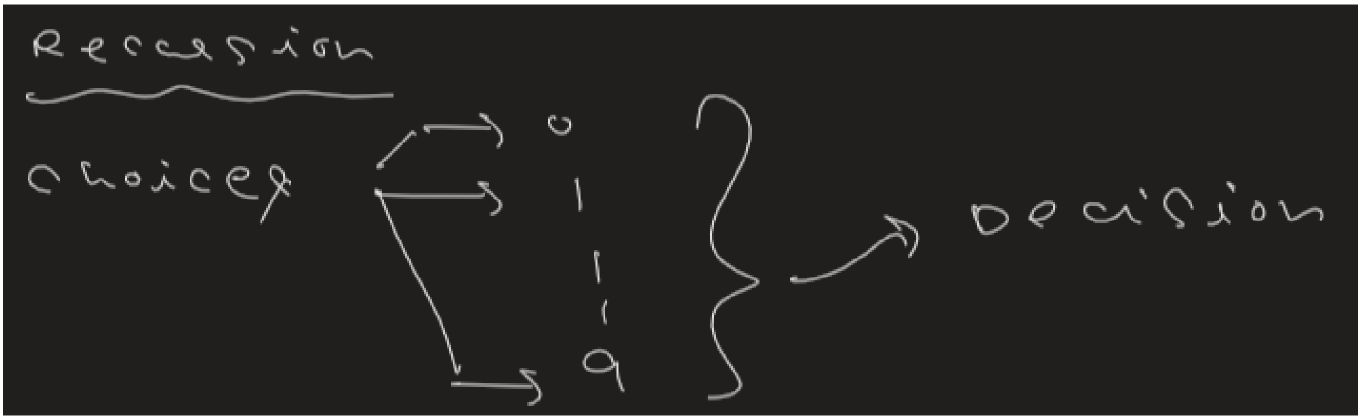
🎯 Lexicographical Numbers

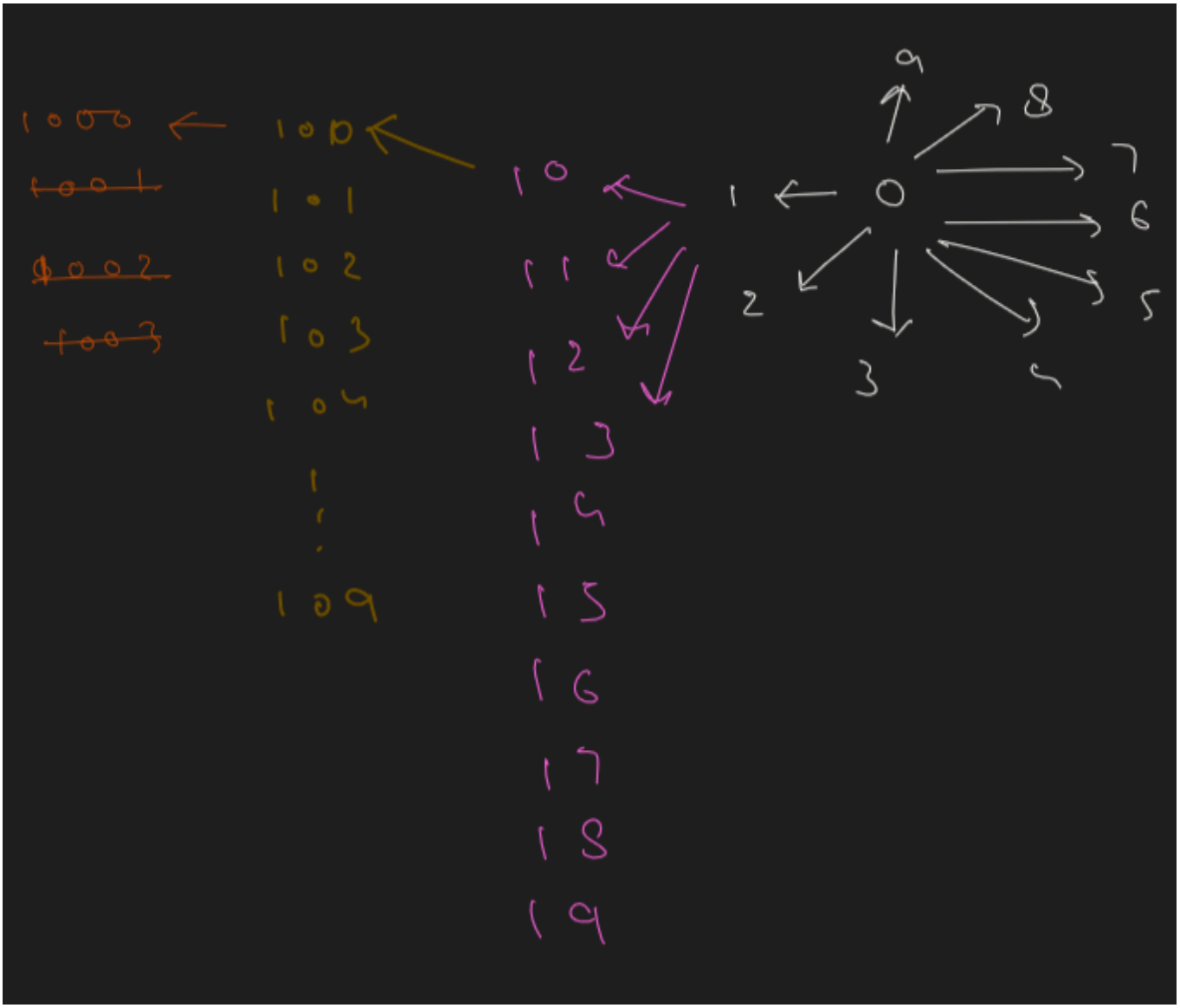
https://leetcode.com/problems/lexicographical-numbers/

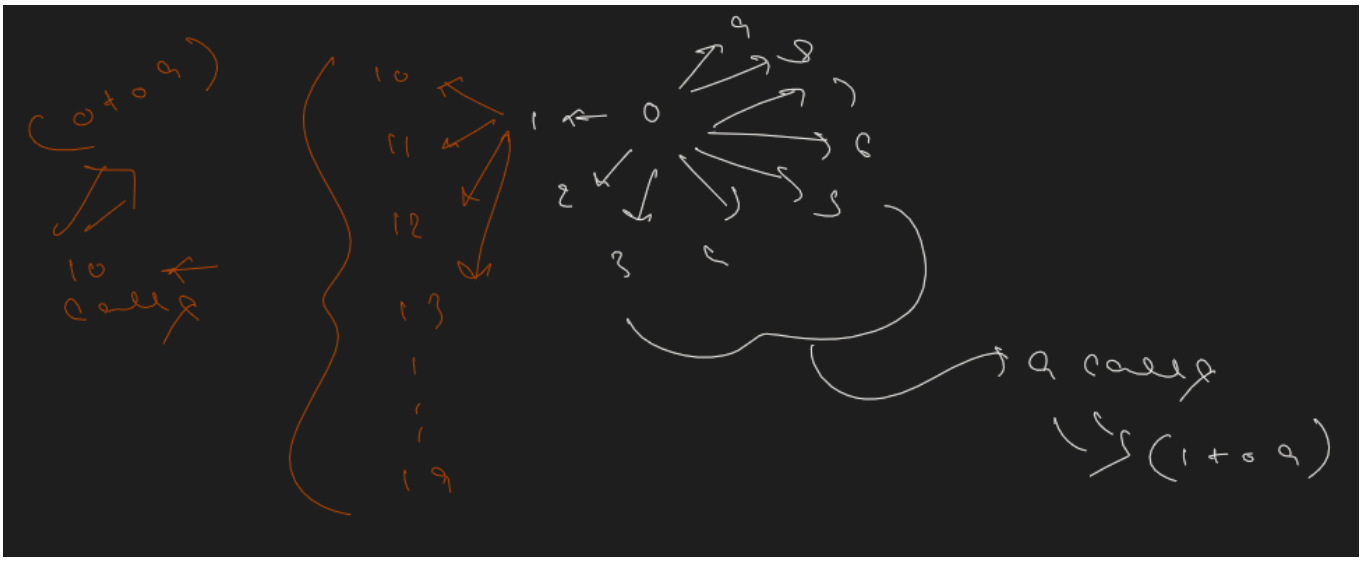


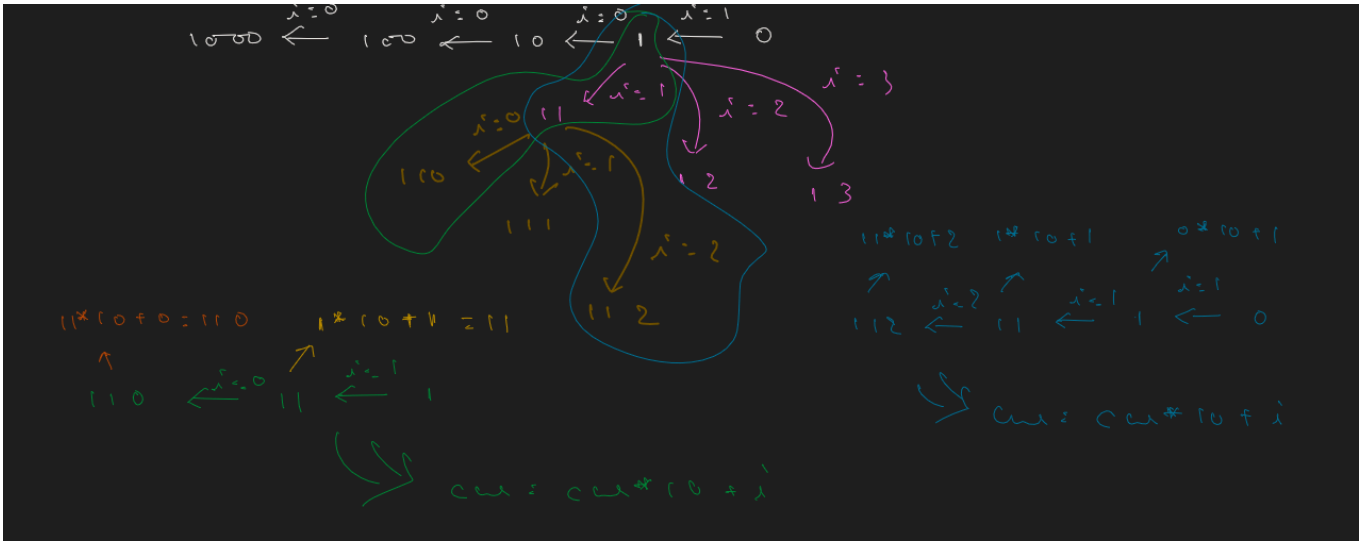

n = 1000
0 1 10 100 1000
101 102 103 ... 109
11 110 111 112 ... 119
12 120 121 122 ... 129
13 130 131 ... 139
14 15 16 17 18 19 ... 199
2 20 200 201 202 ... 209
3 30 300 301 ... 309
9 90 900 901 ... 909


print numbers from 0 to 30
0
1
10 11 12 13 14 15 16 17 18 19
2
20 21 22 23 - - - 29
3
30
4 5 6 7 8 9
0 1 10 11 12 13 14 15 16 17 18 19 2 20 21 22
23 -- -- 29 3 30, 4, 5, 6, 7, 8, 9

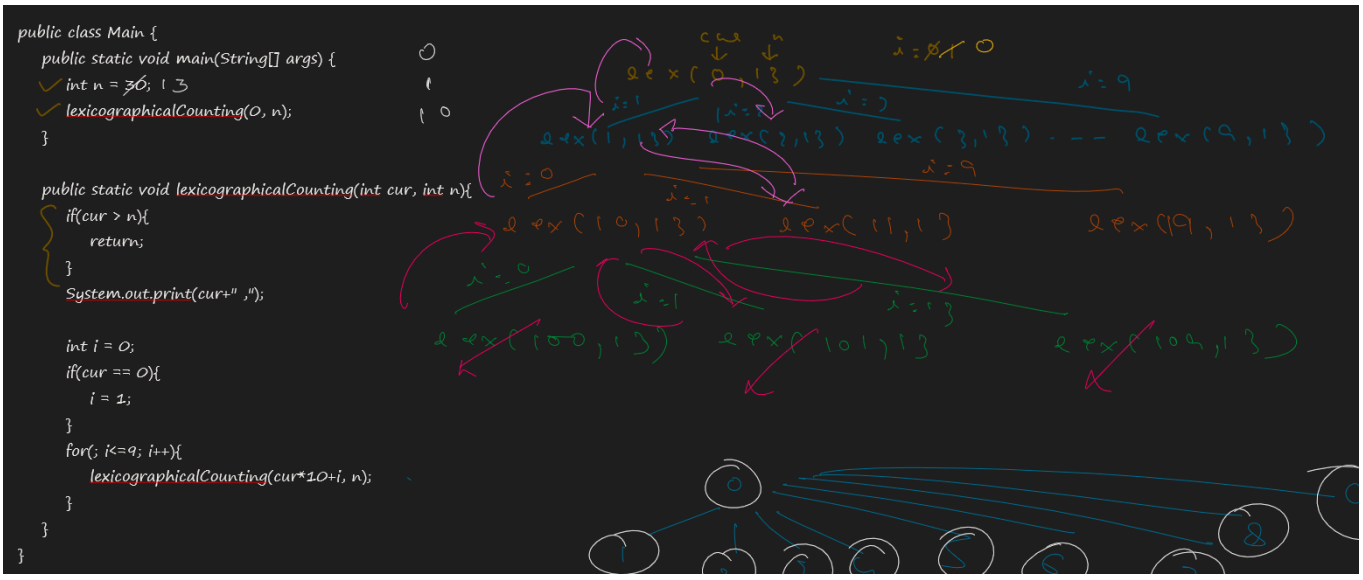

Recursion
choices
0
1
9
Decision

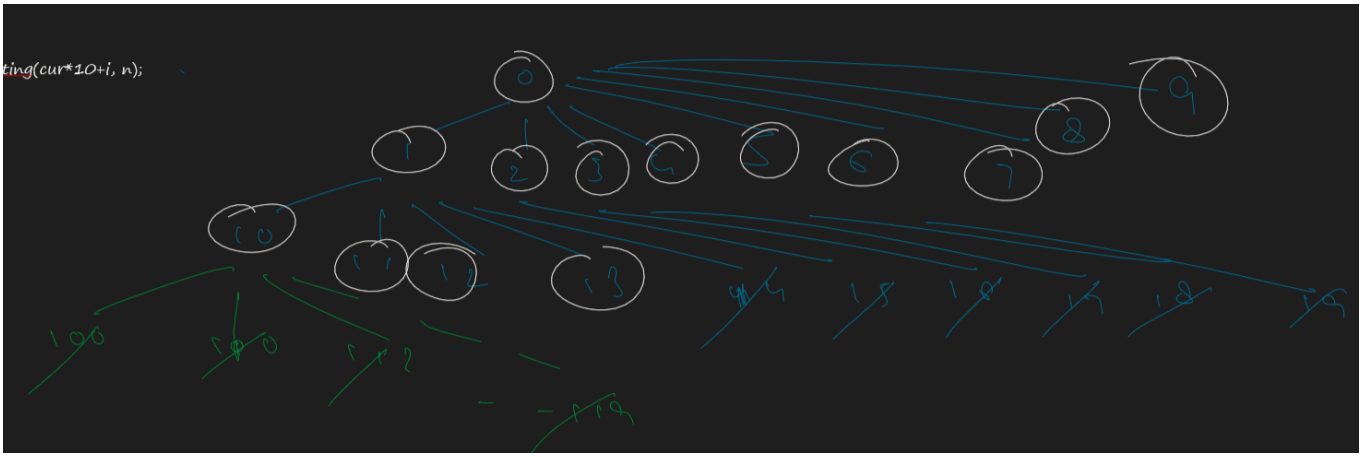

10
11
12
13
14
15
16
17
18
19
0
1
2
3
5
6
7
8
9


10
11
12
13
19
0
1
2
3
9



```java
public class Main {
	public static void main(String[] args) {
		int n = 1000;
		lexicographicalCounting(0, n);
	}

	public static void lexicographicalCounting(int curr, int n) {
		if (curr > n) {
			return;
		}
		System.out.println(curr);
		int i = 0;
		if (curr == 0) {
			i = 1;
		}
		for (; i <= 9; i++) {
			lexicographicalCounting(curr * 10 + i, n);
		}
	}

}
```

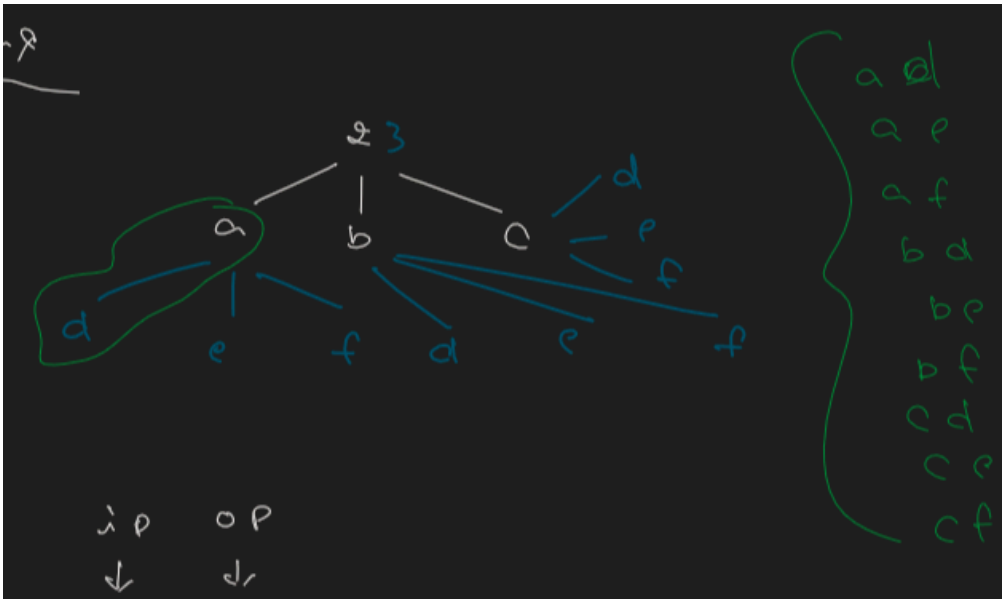


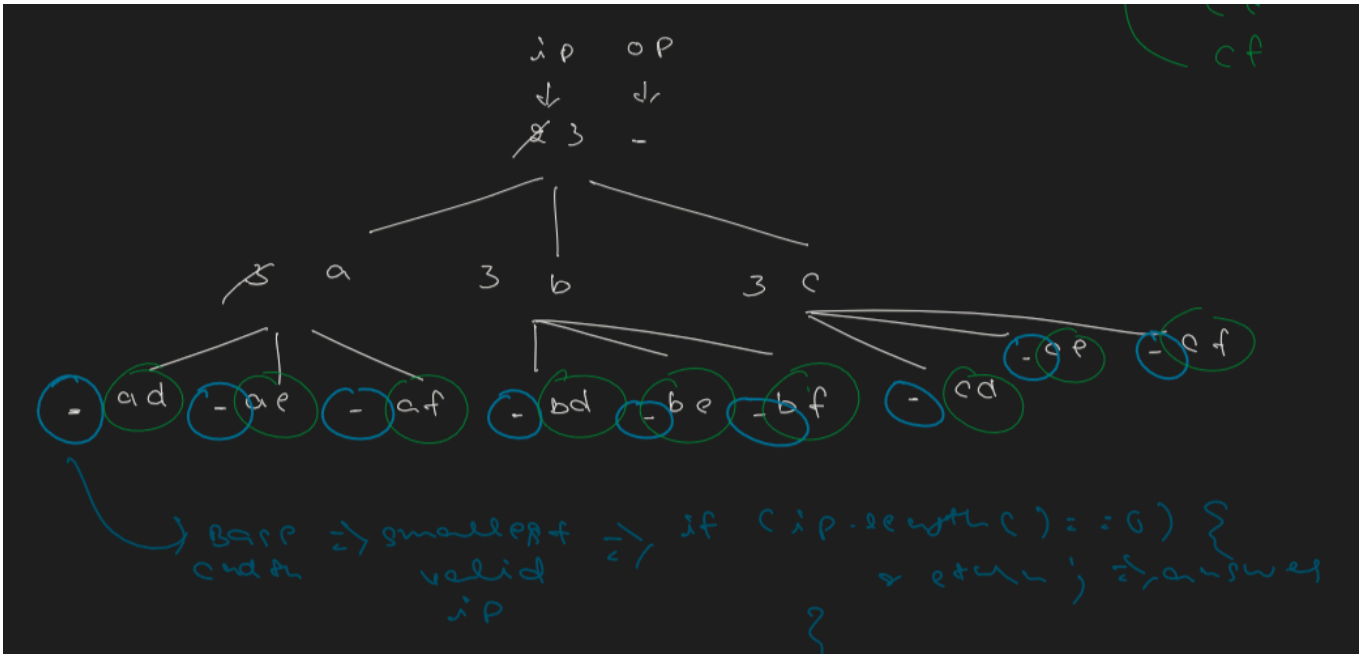


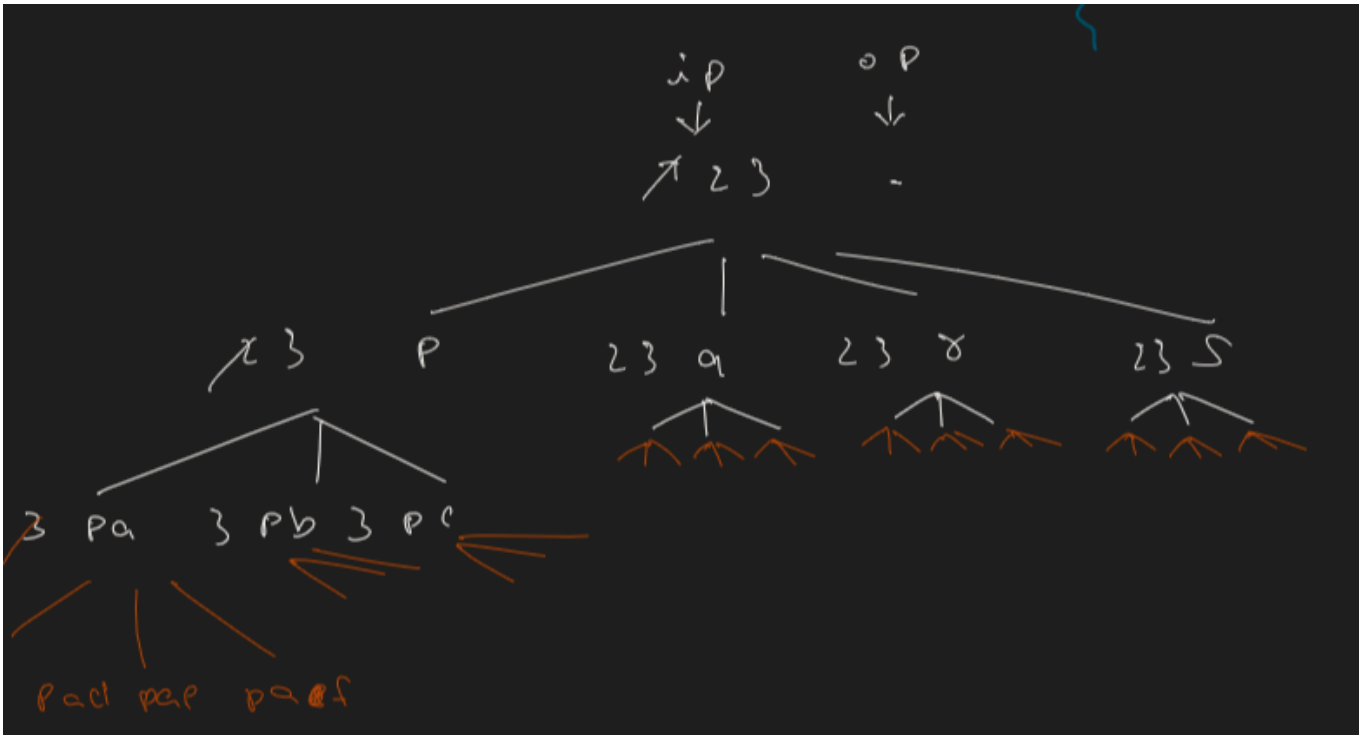
```java
class Solution {
    public List<Integer> lexicalOrder(int n) {
        List<Integer> list = new ArrayList<>();
        lexicographicalCounting(0, n, list);
        return list;
    }

    public static void lexicographicalCounting(int curr, int n, List<Integer> list) {
        if (curr > n) {
            return;
        }
        if(curr != 0){
            list.add(curr);
        }
        int i = 0;
        if (curr == 0) {
            i = 1;
        }
        for (; i <= 9; i++) {
            lexicographicalCounting(curr * 10 + i, n, list);
        }
    }

}
```

🎯 Letter Combinations of a Phone Number

https://leetcode.com/problems/letter-combinations-of-a-phone-number/








```java
class Solution {
    public List<String> letterCombinations(String digits) {
        String ip = digits;
        String op = "";
        List<String> list = new ArrayList<>();

        if (digits == null || digits.length() == 0) return list;

        letterCombinationsUtil(ip, op, list);
        return list;
    }

    static String[] keys = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv",
"wxyz"};

    public static void letterCombinationsUtil(String ip, String op, List<String> list){
        if(ip.length() == 0){
            list.add(op);
            return;
```

```
        }
        String ch = ip.charAt(0)+"";
        String pressedKey = keys[Integer.valueOf(ch)];
        for(int i=0; i< pressedKey.length(); i++){
            letterCombinationsUtil(ip.substring(1), op+pressedKey.charAt(i), list);
        }
    }
}
```

🎯 Grid Path Finder

Objective:

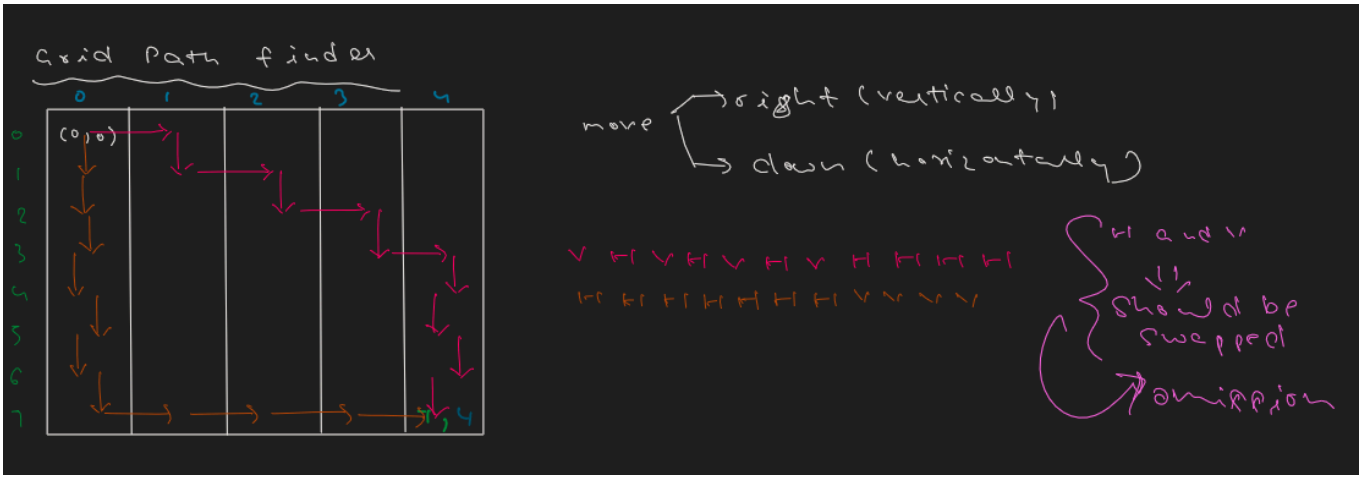
Given a 2D grid of size `n x m`, starting from the top-left corner `(0,0)`, the goal is to print all possible paths to reach the bottom-right corner `(n-1, m-1)` using only the following two moves:

- Horizontal move (H) → move right by one column.
- Vertical move (V) → move down by one row.

Each path should be printed in the order of moves taken, ending with the word "STOP" to indicate the destination has been reached.

Constraints:

- You can only move right or down at any point in time.
- You cannot move out of the grid.
- All paths from `(0, 0)` to `(n-1, m-1)` must be explored and printed.




```
public class Main {
	public static void main(String[] args) {
		int n = 3;
		int m = 4;
		path(0, 0, n - 1, m - 1, "");
	}
	// cr--> current row, cc--> current col, er--> end row, ec--> end col

	public static void path(int cr, int cc, int er, int ec, String ans) {
		if (cr == er && cc == ec) {
			System.out.println(ans+ "STOP");
			return;
		}
```

```
            if (cr > er || cc > ec) {
                  return;
            }
            path(cr, cc + 1, er, ec, ans + "H -> ");
            path(cr + 1, cc, er, ec, ans + "V -> ");
      }

}
```